COLLEGE CODE: 9233

COLLEGE NAME :GOVERNMENT COLLEGE OF ENGINEERING BODINAYAKANUR

DEPARTMENT:COMPUTER SCIENCE AND ENGINEERING

STUDENT NM-ID: 8AE480233C809F1A2D54F980ABF4887C

ROLL NO: 923323104305

DATE: 26.09.2025

Completed the project named as Phase_3

TECHNOLOGY PROJECT NAME : REAL TIME CHAT BOT

SUBMITTED BY,

NAME :THIRUPATHI P

MOBILE NO: 9345110398

PROJECT SETUP:

- Create root project folder named `real-time-chatbot`.
- Initialize backend using `npm init -y` inside the `server/` folder.
- Install backend dependencies: `express`, `socket.io`, `cors`, `dotenv`.
- Set up Express server and configure Socket.io for real-time communication.
- Initialize frontend using `npx create-react-app client`.
- Install frontend dependency: `socket.io-client`.
- Connect frontend to backend via Socket.io.
- Set up basic folder structure for modular code organization.
- Create `.env` files for environment configuration e.g., ports, API keys .
- Verify frontend and backend integration with test message flow.
- Initialize Git repository and push setup code to GitHub.
- Confirm project runs on `localhost` backend: 5000, frontend: 3000 .

CORE FEATURES IMPLEMENTATION:

- Establish Web Socket communication using **Socket.io** on both client and server.
- Implement **real-time message exchange** between multiple users.
- Create a **React-based chat UI** with:
    - Message input box

    - Send button
    - Chat display area

- Store and display messages using **React state**

- Assign temporary **usernames or socket IDs** for message identification.

- Include **timestamps** with each sent and received message.

- Automatically **scroll to the latest message** in the chat view.

- Implement **basic validation** to prevent empty or invalid messages.

- Add support for **bot-generated replies** optional – using OpenAI or static logic .

- Ensure **bi-directional messaging** works across multiple connected clients.

- Allow message sending via **Enter key press** in addition to Send button.

- Optional Display **typing indicator** when a user is typing.

DATA STORAGE:

- Use **React local state** **`useState`** to temporarily store chat messages during a session.

- Optional Integrate **MongoDB** using **Mongoose** for persistent message storage.

- Define a **Message schema** with fields like:

    - `username` or `userId`
    - `message` text
    - `timestamp`

- Connect backend to MongoDB using a connection string stored in `.env`.

- Save each incoming and outgoing message to the database in real-time.
- Retrieve chat history from the database when a user connects or reloads.
- Implement API endpoints for fetching and storing chat messages.
- Handle data validation and sanitization before saving to the database.
- Ensure database errors are handled gracefully with error messages logging.
- Optional Use local storage on the client for offline message caching.
- Plan database indexing on timestamps or user IDs for efficient queries.

TEST CORE FEATURES:

- Test real-time message sending and receiving between multiple clients different browser tabs or devices .
- Verify messages appear instantly on all connected clients without refresh.
- Check that empty messages cannot be sent.
- Test input validation for message length and special characters.
- Confirm that timestamps are correctly displayed for each message.
- Validate that the chat auto-scrolls to the latest message upon new message arrival.
- Test user identification socket ID or username consistency across sessions.
- If AI bot is integrated, verify the bot responds accurately and timely.
- Simulate network interruptions and test reconnection handling.
- Test UI responsiveness across different screen sizes and devices.

- Check for memory leaks or performance issues during prolonged chats.

- Verify proper error handling and display when server or database is unavailable

VERSION CONTROL:

- Initialize a Git repository in the root project folder:

```
git init
```

- Create a `.gitignore` file to exclude:

    - `node_modules`
    - `.env`
    - `build` or `dist` folders

- Stage all files and make the first commit:

```
git add .
git commit -m "Initial project setup"
```

- Create a new repository on GitHub.

- Add GitHub remote repository URL:

```
git remote add origin <your-repo-URL>
```

- Push local commits to GitHub main branch:

```
git push -u origin main
```

- Follow **feature branch workflow**:

    - Create branches for features fixes:
    - `git checkout -b feature real-time-messaging`

    - Commit and push changes regularly.

- Merge feature branches into main via pull requests if collaborating .
- Use **meaningful commit messages** for clarity.
- Regularly pull updates to stay synced:

```
git pull origin main
```

- Tag releases or milestones if needed.
- Use GitHub issues and project boards to track bugs and features.